PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 12/24	A1	(11) International Publication Number: WO 99/07109
		(43) International Publication Date: 11 February 1999 (11.02.99)

(21) International Application Number: PCT/CA98/00738

(22) International Filing Date: 31 July 1998 (31.07.98)

(30) Priority Data:
2,212,251 31 July 1997 (31.07.97) CA

(71) Applicant *(for all designated States except US)*: CROSSKEYS SYSTEMS CORPORATION [CA/CA]; 350 Terry Fox Drive, Kanata, Ontario K2K 2W5 (CA).

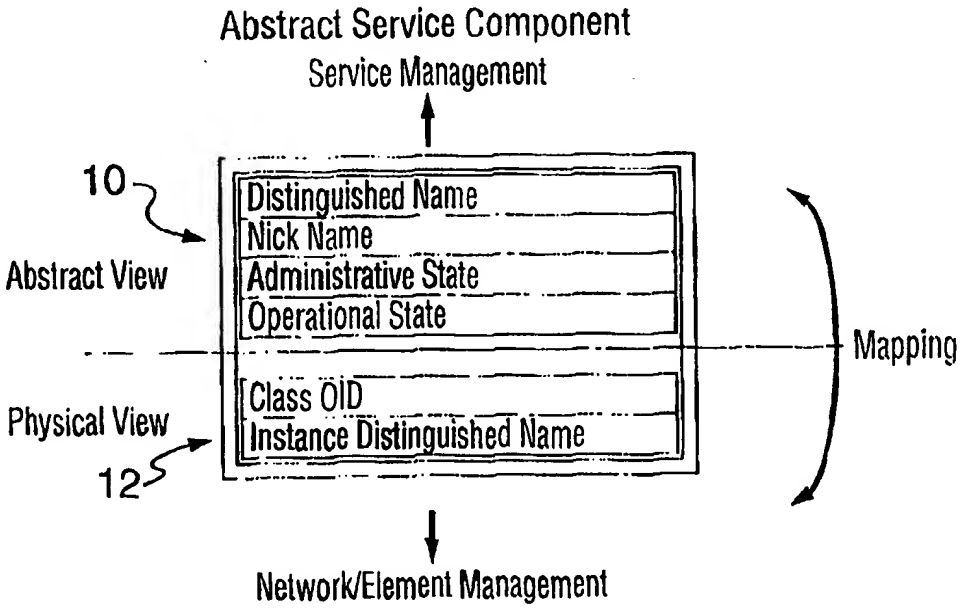
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: BAKER, Steve [CA/CA]; 350 Terry Fox Drive, Kanata, Ontario K2K 2W5 (CA). VINCENT, Bob [CA/CA]; 16 Catherwood Court, Kanata, Ontario K2K 2K1 (CA).

(74) Agent: MITCHELL, Richard, J.; Marks & Clerk, P.O. Box 957, Station B, Ottawa, Ontario K1P 5S7 (CA).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: THIRD PARTY MANAGEMENT PLATFORMS INTEGRATION



(57) Abstract

In a method of importing data into a data processing system from a third party platform, for each physical object an interface object having a canonical form mappable to an underlying, externally invisible form in said data processing system is first created. The interface object has a set of permissible extensions mappable to said externally invisible form. New interface objects can be added by deriving new mappings from said extensions. The data processing system is thus decoupled from the third party platform so that the data processing system can easily interface with multivendor products.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

THIRD PARTY MANAGEMENT PLATFORMS INTEGRATION

Field of the Invention

This invention relates to a system for integrating the service management, for example of telecommunications networks, with third party management platforms.

Background of the Invention

When information is to be exchanged between two systems, for example, between a system monitoring a network and a system monitoring the performance of the network relative to existing service agreements, one must typically agree on a common format and interface. In the case of multiple systems, one must either define a format/interface for each system or impose a least common denominator interface (LCD). The former is expensive to build and difficult to scale beyond a few systems. The latter is restrictive and leads to user dissatisfaction when key functionality is omitted from the interface. The LCD systems fails when a new system is encountered that has limited functionality and is incapable of supporting the LCD interface. In this case, it is necessary either to build a custom interface or adjust the LCD to omit functionality not supported by the new system.

A typical example of a system to which the invention is applicable is the Resolve™™ system from Crosskeys Systems Corporation. This system addresses three primary areas of the Service Management level of the telecommunications management network (TMN) model: Performance Management, Configuration Management, and Fault Management. The applications in these areas are designed to be integrated and work together in configurable combinations through the use of common core components.

A customer has one or more services and one or more contracts with the service provider. A contract includes a definition of the specified quality of service (as captured in a Service Level Agreement). A service profile defines a certain quality of service for a range of services. For example, a "Gold!' service profile may specify 99.995% availability whereas a "Silver service profile may specify 99.9% availability.

Each Service is supported by one of more Service Components. A service component could be a physical piece of equipment such as a termination point a logical entity such as

an end-to-end path or an arbitrary external system such as a customer support desk. All service components support the notion of availability, and most support the production of performance measures.

This object model is maintained in the Service Management Information Base (SMIB). The SMIB, is an operational data store. In addition to customer, contract and service information it contains current, volatile data (such as path states, and events), which may be used in real-time reports.

The Resolve™™ system also maintains two other major information bases, the Historical Information Base (HIB) and the Summarized Information Base (SIB).

The HIB contains detailed, time-varying data (such as path statistics, and old state and event data). This data is held in the HIB for a short period of time (typically 60 days). This data can be used to produce detailed reports.

The SIB contains summarized information, based on the data in the HIB. Data in the SIB may be up to 180 days old. Most relevant reports will be run against the SIB, as its information is the most meaningful. The SIB information may be used to identify trends and patterns in usage and availability.

Availability and performance data is captured from a number of information sources and used to update the SMEB and the HIB. Periodically the HIB is summarized to produce one or more SIBs. Users can then define and produce reports, as well as configure existing and new services.

In order to provide product service management functions in the telecommunications domain, the Resolve™™ system must interface with a wide variety of network management systems. The capabilities and strengths of the third party management platforms vary. The range of applications and customers also vary from platform to platform. Simply aiming at a lowest common denominator approach to integration will lose many of the advantages of a particular platform. Many different interfaces must therefor be provided.

An object of the invention is to alleviate this problem.

Summary of the Invention

According to the present invention there is provided a method of importing data into a data processing system from a third party platform collecting data from physical objects, comprising the steps of creating an interface object having a canonical form with a set of permissible extensions, mapping said canonical form to an underlying, externally invisible form existing in said data processing system, creating additional interface objects using said permissible extensions to derive new mappings for said additional objects, associating said interface objects with said physical objects, converting said data to be imported to said canonical form in said interface objects, and transferring said data through said mappings to said data processing system, whereby said data processing system is decoupled from said third party platform.

There are thus three fundamental data formats in accordance with the invention. The imported data, which is present in the third party form, the intermediate format, which is of canonical form, and the target form, which is externally invisible. The externally invisible form is an abstract data format that supports arbitrary attributes on arbitrary objects. The collected data may, for example, be availability and performance information, which is used to perform a variety of service management tasks, but the invention is equally applicable to any data that needs to be imported into a data processing system that is adaptable to multiple vendor platforms.

In accordance with the invention, if an existing interface object and associated mapping exists and it is desired to import something similar, but different, the mapping for a new interface object can be derive by expressing it as an extension of an existing mapping. The derived mapping contains only the differences between itself and the original mapping (the one it derivedfrom). It is not necessary to define the mapping from scratch.

A platform in this context is any kind of underlying computer or data processing system on which platforms can run. Different platforms may have different operating systems and data formats.

A service model may also be exported to the third party management platform to generate alarms and/or trouble ticket information relating to this object model.

In the method according to the invention, a Gateway application is run on the management platform that will exchange information between the system and the platform.

The gateway interacts with the platform using the locally specified application programming interfaces. From a user perspective, the gateway will appear, to the extent that it is visible at all, as another platform component, consistent in design and interface with all other components.

The interface is specified by an object model. A base object model is presented, and a range of possible extensions to that model are then outlined. Thus the interface can vary in complexity and sophistication depending on user/application need.

The design provides for system integrators to customize the behavior of the gateway, using a Turing complete language, to enable local modifications to be made quickly and easily.

The invention also provides a system a system for importing data into a data processing system from a third party platform collecting data from physical objects, comprising an interface object for each physical object, said interface object receiving said data and having a canonical form and having a set of permissible extensions mappable to an underlying, externally invisible form in said data processing system so as to decouple said data processing system from said third party platform and permit mappings for new interface objects to be derived as extensions of existing mappings.

Brief Description of the Drawings

The invention will now be described in more detail, by way of example, only with reference to the accompanying drawings, in which:-

Figure 1 shows a service component object;

Figure 2 shows the mapping between object models;

File 3 illustrates file format conversion;

Figure 4 shows as scan notification report;

Figure 5 shows the interfaces relating to the exchange of service information with a third party management platform;

Figure 6 shows a COBRA interface;

Figure 7 shows the feature architecture of an interface;

Figure 8 shows a network-interface operating environment;

Figure 9 shows the components of a network interface;

Figure 10 shows a two-step translation process;

Figure 11 shows a multivendor parser; and

Figure 12 shows an event collector.

Description of the Preferred Embodiments

The invention will be described with respect to the Resolve™™ service level management agreement system referred to above. The objective is to provide a generic interface that permits the Resolve™ system to interface with multivendor third party platforms without the need to re-write internal software.

Figure 1 shows an interface object or service component that might exist on a third party network management platform. It has two halves, an abstract half 10 that is visible to the service management system and a physical half 12 that references the object in the service level management software that denotes the physical network facility.

Such a service component is intended to be applicable in a wide variety of cases. All concrete service components are viewed as particular types of this more general service component, which termed an *Abstract Service Component (ASC).*

The ASC object monitors the network object for changes in availability. The ASC object therefore acts as a form of proxy object. Changes are then reflected in the ASC (as a change in the operational state) and an event is forwarded to service management system. For every ASC object there is exactly one network object.

To use a network object as a service component it must support some notion of operational availability. For objects that support such a notion in a standards compliant

manner (i.e. they have an Operational State attribute modeled according to X.721) the implementation of the ASC is simple.

However, it is possible that some network objects may use some other names for operational state or identify changes in availability by some other means. A configurable mapping is thus provided between the operational state of the ASC and the equivalent state in the network object. This is achieved by using a mapping language to express the relationship between-the ASC and the network object.

An attempt to perform an action on the ASC is mapped into an action on the network object. This mapping is performed by embedding a flexible scripting language within the ASC. Actions on the ASC result in a script being executed. Thus a request to receive event state changes result in a script being executed that in turn gathers an event from the physical object and interprets it appropriately.

For example, suppose a network object representing a multiplexer modeled operational state as an attribute called "available". Suppose further that the attribute took the values "'yes" and "no", but no event was issued when the attribute was changed. It would be possible to associate with the request to receive operational state event changes from the ASC a script that polled the multiplexor object and changed the ASC's operational state attribute and issued an event change when appropriate. Thus Resolve™ can perform availability based reporting on an object that a) does not have an operational state and b) does not issue state change events.

The mapping between object models is show in more detail Figure 2. An arbitrary network object 20 on a third party management platform is mapped to an object 22 in the service level management system, in this case the Resolve™ Service Model. The mapping is achieved via an ASC 24. The ASC decouples the service level management system from the management platform without loss of functionality. This enables support for new management platforms to be added without requiring changes to the service level management system which would necessitate a new release and necessary longer development times.

As the ASC 24 runs on the remote platform it must conform to the relevant platform API. This implies that although the interface the ASC 24 presents to the service level

management system is constant, the implementation of the ASC will be necessarily platform specific.

In addition to the availability of network object, there may be additional, largely static, information that might be usefully passed to the service level management system, such as the Available Bit Rate of an ATM PVC. The attributes of the ASC object can be extended to include this additional information.

There are three main options for the ASC:

• Fixed number of fixed-name attributes

Add a fixed number of attributes called *attribute value', attribute-name', attribute_yalue~ ..., attribute. name"*. The ASC would relate these general purpose variables to particular variables in the network object, and would use the *attribute-name'* attribute to store a meaningful presentation name to be used by the service level management system. These attributes would be of type to accommodate a wide variety of options.

* Variable number of fixed-name attributes

Add an array of attribute/value pairs. This is a a variation on the arrangement above, but permits a variable number of attributes.

• Variable number of variable-name attributes

This is the most flexible option permitting user-specified additions to the object model. Such additions could be performed by directly modifying the specification of the object (in GDMO - Guidelines for the Definition of Managed Objects- , for example) or by using a platform specific tool that permits such operations. Note that this option requires that the ASC implementation and the Resolve™ interface be data driven.

It is possible to generalize the notion of an ASC, containing a number of arbitrary attributes, to one capable of collecting statistical information from the network management system. Statistical information is that information that is typically reported at a given interval (for example, every 15 minutes). Such information is typically quite large and issuing attribute change events quickly becomes unrealistic and therefore a more considered approach is required.

Performance information is traditionally presented in a flat file, usually consisting of ASCH data. More recently, the standards bodies have specified an approach using OSI-events and some vendors now support a native OSI interface to performance data.

In many cases the statistical information can be obtained directly in a flat file and this is preferred as it is simpler. Performance information is presented in a native format and must be converted into the format maintained by the service level management system as shown in Figure 3. The performance data attributes are based on the relevant standards (ATM Forum, Frame Relay Forum, etc), the same standards used by the equipment vendors. Thus conversion is largely a matter of accommodating differences in presentation, rather than in content.

In some cases a vendor may omit a standard parameter, supply a useful but non-standard performance parameter, or calculate a standard parameter in a non-standard way. In this case the conversion mechanism becomes more complex and a toolkit solution is required.

From the perspective of events and object models, the ASC provides a decoupling of the service level management system from the management platform. This decoupling is also desirable for importing performance data from file and therefore the conversion is performed in two stages. The native format is first mapped to a canonical form which is then read by a statistics importer. The canonical form is an abstract data format then supports arbitrary attributes on arbitrary, objects.

Thus, there need only be on statistics importer and adding support for a new native format does not require changes to Resolve™. The conversion to canonical form is typically a straightforward task and can be performed by a systems integrator, platform vendor or by CrossKeys. This conversion can also be applied to binary files which are often used as a more compact format for performance data. For ASCH files a wide variety of parsing/translation tools can be used, for example yacc/lex or CrossKeys' own parsing toolkit, AMPERE.

There may be cases where statistical information is available only via the network management system. One attractive approach is to use Q.822, as is done in CrossKeys Traffic Management applications (Altus~m). Q.822 provides a generic framework for collecting performance information. Objects that have attributes that change over time,

are called *monitored objects. To* collect information from monitored objects at a certain reporting interval an object called a scanner is used (scanner objects are defined by X.738). A scanner object gathers many attribute values together and emits a *Scan Report Notification* that is, in essence, an optimized form of attribute value change event used when the number of attributes, and frequency of change, is high.

When information is available in the form of a Scan Report Notification it could either be sent directly to Resolve™ or converted into a flat file format before importing. The format of a Scan Report Notification is well-defined and therefore the need for a vendor-specific format conversion phase is removed.

A Scan Report Notification is an event and therefore the Abstract Service Component model can be used again, as shown in Figure 4. The Resolve™ Statistics Collector receives Scan Report Notifications from an arbitrary network component, via the ASC. As the figure suggests, the fundamental mechanism for relaying events remains the same.

Scan Report Notification can be seen as a type of canonical form. Performance data received from the arbitrmy network component can be in any form, and it is then mapped to canonical form (Scan Report Notification) and then relayed to a statistics collector.

The intent of an USC is that it be general purpose. Thus if a network object model supports a termination point object and a link object, then both would be represented in Resolve™ as an instance of an USC. Obtaining the operational state of the termination point object may be quite different from that of a link object. Thus in defining the mapping it is necessary to know to what type of object the ASC is pointing. In essence, type information is required to support polymorphic behavior of the ASC.

This type information is also of interest to Resolve™. One of the value-added capabilities of Resolve™ is a suite of standard reports. Such reports are based on a detailed understanding of the technology. If the service component is of type "ATM VPC" then a number of standard reports are available, for example.

With an ASC one cannot know a priori what technology it will be supporting and thus the possibility of value-added service disappears. However, by defining some essential characteristics of, say, an SDH Add-Drop Mux (ADM) then it would be possible to pre-define some SDH ADM reports. Such reports might require that information about X,

Y, Z be available. If such information is available it would be desirable if there was some way to tell Resolve™ that the ASC was in fact being used as a proxy for an SDH ADM and that parameters X, Y and Z are available and that the standard reports were applicable.

One could envisage that as a number of third party applications were integrated then the possibility of developing a suite of standard reports for each technology type would increase.

Note however that the concept of type would need to be quite sophisticated. If a SDH ADM supports parameters X, Y and Z then a suite of standard reports is possible. If a SDH ADM supports parameters UV,W,X,Y * and Z then a super-set of reports could be supported. If a SDH ADM supports parameters U, V, X and Z, but not Y, then some smaller combination of reports is possible.

Rather than develop a complex type system, the object ID (OID) of the network component object is used as an indication of type. An Object ID is an OSI-specified means of uniquely identifying an object class. All OSI object classes have a unique Object ID making it an excellent choice for distinguishing between classes of objects without inventing a complex type scheme. This OID can then be used to control both the polymorphic aspects of the mapping and as a way to identify to Resolve™ the capabilities of the network object and hence the extent to which preexisting reports would be applicable.

The ASC objects act as proxy objects for the real network objects and as such should not be directly visible to the end user. To make this possible, it would be desirable if each time a physical service component (PSC) object was created, an ASC would be automatically created. Similarly for object deletion.

It is simple for a systems integrator to write scripts that could auto-discover PSC objects and create the equivalent ASC objects. Such a script must be able to distinguish between different types of PSC objects so that appropriate mapping could be employed.

The use of an Abstract Service Component enables information to be gathered from an arbitrary network component object and incorporated within Resolve™ for the purposes

of Service Management. An extended Abstract Service Component allows more complex information to be exchanged, including performance data.

By mapping the network object model to the Resolve™ object model in two stages (i.e. first to an Abstract Service Component) the integration effort can proceed a greater rate. No changes to Resolve™ are necessary and, by permitting the ASC to be modified without recompilation, the integrator can quickly bring Resolve™'s service management capabilities to a new type of network object.

The same event mechanism can be used to relay performance data in the form Scan Report Notifications. Such a solution is both standards based and compatible with CrossKeys Altus products.

In situations where performance data is available in the form of a flat-file, an alternative approach is adopted. Performance data is first mapped to a canonical form. The Resolve™ statistics collector then reads this canonical form, instead of the native format. This permits support for new native formats to be added without requiring changes to Resolve™.

The two solutions to importing performance data can be seen to be logically equivalent, differing only in the exact definition of the canonical form (ASCH text in a flat file vs Scan Report Notification). This equivalence, and the previously noted relationship to the model shown in Figure 2, means that a single, uniform architecture can be employed to solve a range of integration issues.

All these mechanisms (which are essentially variations on a common solution) address a common objective; to support the integration of new information sources without requiring continual change to Resolve™. By avoiding changes to Resolve™ the integration with other platforms is de-coupled from the Resolve™ release cycle, allowing faster integration. By making this integration configurable and open, it is possible for it to be performed by third parties.

Another feature of the described system is the ability to exchange service information with the third party platform. This is achieved by

- making Resolve™'s customer, service and service component objects visible on a third party platform,
- exporting Resolve™-detected SLA violations into the third party platform's alarms and trouble tickets facilities, and,
- receiving trouble tickets as a means of supporting customer-perceived outages.

In one embodiment, the Resolve™ Object Model is exported in the form of a flat file, enabling third party platforms to incorporate this information in a variety of applications.

In a more sophisticated embodiment, an object model is defined on the remote platform (using GDMO, for example) and implementing a proxy-agent. Attempts to show the attributes of a service entity, for example, would result in a request being sent to Resolve™ to get the real attributes and then returning them to the platform.

If the proxy-agent did not cache any of the attribute values then there is no need to synchronize with Resolve™ with respect to Resolve™-initiated attribute value changes. It will, however, be necessary to synchronize with respect to object creation/deletion. . This could be done once on start-up and subsequently tracked via Resolve™-generated object creation/deletion events.

When a SLA (Service Level Agreement) is violated a Quality of Service alarm is raised. Such an alarm is raised against either the service or the service component. The format of the alarm is well known and raising it on a third party platform is comparatively straightforward.

It adds significant value to the SLA violation alarm if the network component that had caused the violation is identified. In cases where it is possible to identify the component, one could raise the alarm against the network component directly, or continue to raise the alarm against the service component but somehow include in the alarm the identity of the network component. In both cases it is necessary to identify the network component by name. This information is stored in the ASC, and is therefore readily accessible.

To include such information in an OSI alarm is straightforward. It could be included in the Additional Information or Additional Text fields. The latter is simpler, but less flexible.

To raise the alarm against the network component itself is potentially more complicated. Such an object will have a GDMO (or similar) specification that describes what events it is potentially capable of generating. If this description did not include a QoS alarm, it would be necessary to augment the specification.

Exporting Trouble Tickets is similar to exporting alarms in that it requires that the objects referenced in the TT (e.g. service) be known to the remote platform. TTs can be created by executing scripts directly on the platform. These scripts can be user-defined, enabling Resolve™ to support a variety of creation schemes.

However, it is envisaged that the prime task of Resolve™ is to raise alarms indicating actual or pending SLA violations. Whether these violations should result in a TT, and how the TT is populated, is usually a local decision influenced by local policy as much by technology, and therefore Resolve™ would not typically create TTs directly.

The intent here is to allow an external trouble ticketing systems to report changes in the availability of a service. When a customer reports a service outage he/she is unlikely to identify the failed service component; rather the customer would report that the service as a whole was unavailable. This could result in a trouble ticket indicating that the service is (perceived to be) unavailable. When Resolve™ receives a trouble ticket, it will mark the corresponding service or service component to indicate a trouble ticket is active. The outages are calculated when the trouble tickets are closed.

Importing Trouble Ticket information into Resolve™ is performed by means of a CORBA-based importer. Thus from a remote platform perspective there are three tasks to perform.

- Gather information about the creation/modification/deletion of Trouble Tickets,
- Filter out TTs not relevant to Resolve™,
- Export TTs in a format understood by Resolve™.

At a minimum the person creating the trouble ticket needs to know the names of all the services and service components (and by implication the names of the customers). Such information is provided in the exported Resolve™ object model.

An application would need to run on the remote platform and gather object creation/delete and attribute value change events pertaining to trouble ticket objects. It is relatively straightforward to filter on the managedObjectInstance attribute of the Trouble. Ticket and then use the capabilities of the Resolve™ Trouble Ticket importer to transmit the information to Resolve™.

Exporting the object model as a flat-file, while simple, is always be a desirable feature as it permits the data to be used in arbitrary ways, from sophisticated network management applications to simply incorporating the data in a spreadsheet.

A more sophisticated level of interface is provided by implementing a proxy agent on the remote management platform. This will permit the Resolve™ objects to be used in the same way as any other object on the management platform, including being displayed on graphical maps or referenced in trouble tickets. Users of the management platform would be unaware that the objects were in fact managed by Resolve™.

Importing trouble ticket information back into Resolve™ is performed via a predefined CORBA interface, permitting simple gateway applications to be developed on the management platform.

A suitable method for transmitting information between Resolve™ and the third party platform is to use a COBRA interface. Other methods, such as Q3, can be employed. COBRA stands for Common Object Request Broker Architecture. CORBA offers the greatest chance of providing a uniform transport mechanism across a range of platforms and is therefore discussed in greater depth.

The CORBA interface can itself be broken down into the three possibilities shown in Figure 6. A common intermediate representation will be used to exchange information as this provides a high level interface that will be consistent across all management platforms. Adding support for a new management platform requires only that the platform specific component be rewritten, enabling a new management platform to be added without a change to Resolve™ that would require a new release and unnecessarily delay integration.

Moreover, this interface can grow in scope making more the system functionality available, permitting more sophisticated applications to be developed on the third party management platform.

The interface with the third party platform will now be described in more detail. The Multi-Vendor, Multi-Platform Feature as defined has two basic groups of requirements;

- Provide a capability to easily build network interfaces that provide support equivalent to existing network interfaces. This capability must be provided in such a way that it can be extended in the future to support additional network interface requirements

These requirements are translated into a simple three-layered architecture as illustrated in Figure 7. The translation framework 30 provides the support for general purpose translation systems and is intended to address the second type of requirement, namely extensibility.

The translation library 32 is a suite of generic translations that aid the process of building network interfaces. It is possible to build network interfaces using only the translation framework 30, but there is still the opportunity to factor our generic work and thereby simplify the task of adding new interfaces.

The Vendor specific work 43 represents the additional configuration work that is necessary to build a network interface for a specific vendor's equipment. The vendor specific work is build upon the translation library, but as the Figure illustrates this is not necessary (although it is likely to involve more work).

A Network Interface (NI) performs the following tasks;

- Event Collection: The collection of events, in near real-time or in batch mode, pertaining to network objects, translation into the Resolve™ representation and storage in the SMIB.

- Object Synchronization: The uploading of objects (paths, path endpoints, etc.) from the element or network management system, conversion to the Resolve™ representation and storage in the SMIB.

- Statistics Importing: The importing of statistics for network objects, conversion to the Resolve™ representation and storage in a file for subsequent loading into the HIB.

A network communicates directly with the element/network management system to collect information and translate it to a standard form. The E/NMS communicates this information to the Resolve™ Server.

The behaviour of the network interface is highly E/NMS dependent. Support for configuration is provided in the form of "mappings" which define how the network interface should translate, or map, information from the E/NMS to Resolve™. The network interface will typically make use of an E/NMS library of routines to access vendor/platform specific data. The mapping then performs the conversion of native information and relays the information back to the Resolve™ Server.

Figure 8 illustrates the idea by showing the flow of information from the network element 40 through the E/NMS 42 into the network interface 44 where it is then translated.

Figure 9 shows the relationship between the network interface shown in Figure 8 and the translation framework by illustrating the components that are used to construct a specific interface.

The Multi-Vendor, Multi-Platform feature supplies;

- the "Translation Framework"
- a library of vendor independent mappings (described belowe)
- A generic interface to E/NMSs that present data in flat-files
- A skeletal event collector with the ability to dynamically load vendor specific event collectors.

Using this approach it is possible to build a configurable multivendor network interface that can perform event collection, object synchronization and statistics collection. Such a network interface is then configured to meet the specific needs of E/ g. To build a specific network interface (e.g. to Ascend Frame Relay equipment) typically requires;

- some vendor specific mappings

- possibly an E/NMS interface (if communicating via flat-files is not enough)
- some additional simple translation utilities (if the flat-file library is to be used)

A further explanation of mappings, and how they can be used to perform event collection, object synchronization and statistics collection, is given below.

Figure 10 illustrates the two-step approach to interfacing with a third party platform in accordance with the invention. The native format, be it file-based, or otherwise is converted in an intermediate format. This translation is primarily syntactic in nature. This intermediate format is then translated, by a "mapping", to the target format. This translation is typically concerned with semantic translation.

As the native format is clearly E/NMS specific and therefore so is the translator that translates this to intermediate format. The mappings are also E/NMS specific (although not as much as might be imagined at first) and therefore the construction of a network interface to a new type of E/NMS will typically require the creation of a translator and some mappings.

The rationale for a two-step translation is as follows;

1. Two forms of translation must be performed - syntactical and semantic. Syntactical transformation is much simpler and thus it is desirable to retain this simplicity by separating it from the complexity of the semantic transformation.
2. Two-step translation is a tried and trusted approach in compilation theory. The translation required for MV interfacing is a simplified form of the same problem faced by a compiler.

E/NMS data is imported in a two stage process. E/NMS data is first converted into a technology independent file format, called the Resolve™ Object Format, and then it is mapped to a Resolve™ object.

The Resolve™ Object Format consists of Resolve™ Canonical Objects (RCO). Each RCO belongs to a named class, and has a (possibly empty) list of attributes, called Resolve™ Canonical Attributes (RCAs) and a (possibly empty) list of contained RCOs.

A RCA has a name and a value. No two RCAs in the same RCO may have the same name.

Nested RCOs may be used to express containment relationships. A nested RCO inherits the attributes of its containing RCO. Attribute overriding is possible, with the most deeply nested value taking precedence. The effect is similar to the lexical scoping rules employed in most programming languages.

RCOs and RCAs may be represented in textual form, in the form of a ROF file. The definition of a ROF file, using an Extended BNF notation, is as follows;

file ::= <header> { <RCO> }*

header ::= **ROFV1.0**

RCO ::= **class** <name> '{'{<RCA>}* { <RCO> }* '}'

RCA ::= <name> = <value> ;

name ::= <string>

value ::= <integer> | <real> | <string>

C++ style // comments may be used in a ROF file.

A ROF File is a header followed by sequence of zero or more Resolve™ Canonical Objects (RCOs).

The class name of an RCO is used to identify a *mapping*. A mapping is an object that embodies the required semantic transformation of an RCO to a given format.

Note that there is a class name, but no instance name.

The following examples illustrate some of the concepts identified above.

Thus the following is illegal because *attribute1* is defined twice;

```
ROFV1.0
class A {
        attribute1 = "foo";
        attribute1 = 2;     // illegal - redefinition of attribute1
}
```

The following example shows a legal example of a nested RCO and attribute1 will have the value of *bar* when the class *B* mapping is executed;

```
ROFV1.0
class A {
        attribute1 = 1;
        attribute2 = 3;
        class B {
                attribute1 = "bar";    // legal - overrides previous definition.
                attribute2 = 2;        // also legal
        }
}
```

Note that two RCOs may have the same class name (in fact this is quite common), but they are not required to have the same RCAs. Thus the following is legal;

```
ROFV1.0
class A {
        name = foo;
        attribute1 = 1;
}
class A {
        name = bar;
        attribute2 = 1;
}
```

There is no specific mention of time in the ROF syntax. However time information can be readily included. The advantage of not including time as mandatory ROF syntax is that it does not force a particular solution on to the mapping implementer. A common technique is to take the time from the system clock so a mandatory requirement to include time information would be unnecessarily restrictive.

The disadvantage of not including time in the ROF syntax is that puts the onus on the mappings to check for a syntactically correct time. However, this can be mitigated by use

of inheritance in the mapping implementation language which would permit such validation and subsequent processing to be implemented once and re-used many times.

In the following example the object with a class name of *fr_stats* implicitly has an attribute called *timestamp*.

```
ROFV1.0
class time {
        timestamp = "1997-11-11 12:30:05";
        class fr_stats {
                a = 1;
                b = 2;
        }
        class fr_stats {
                a = 2;
                b = 3;
        }
}
```

From the perspective of the *fr_stats* class, the following example is equivalent to the previous example;

```
ROFV1.0
class fr_stats {
        timestamp = "1997-11-11 12:30:05";
        a = 1;
        b = 2;
}
class fr_stats {
        timestamp = "1997-11-11 12:30:05";
        a = 2;
        b = 3;
}
```

The ROF format outlined in the previous section captures only the syntactic aspects of translation. It is easy to imagine how a file from a E/NMS may be (syntactically) transformed into a ROF file, but it is less clear what purpose this might serve as the more challenging problem of semantic translation is not addressed.

Semantic translation is performed by objects called *mappings*. The class name in an RCO identifies the mapping to be used on it. A mapping is applied to an RCO. The following example illustrates how mappings are related to RCOs;

```
ROFV1.0
class fr_stats {
        filename = "stats1.dat";
        a = 1;
        b = 2;
}
class ATM_stats {
        filename = "stats2.dat";
        X = 3;
        Y = 4;
}
```

This can be interpreted as follows - find a mapping called *fr_stats* and execute it, passing as parameters the values of the RCAs of the first RCO (namely *filename*, *a* and *b*.) And then, find a mapping called ATM_stats and execute it, passing as parameters the values of the RCAs of the second RCO (namely *filename*, *X* and *Y*).

Note that mappings are applied to individual RCOs, not to complete ROF Files. A ROF File is a way of defining a collection of RCOs, each of which in turn identifies the mapping that should be applied to it.

Quite what the mapping actually does is not important (to this explanation). It would be possible to write a mapping called *fr_stats* that took the values of *a* and *b* and added them together and wrote the result to a file called "stats.dat" (i.e. the value of *filename*).

Thus the implementation of a mapping becomes the means by which semantic translation is expressed. As will be discussed in later sections, mappings are implemented in a

conventional programming language (C++ and Java in this release) and may be as simple or as complex as is needed to express the required translation.

The Multi-Vendor Parser is a particular use of the translation framework that will permit the development of network interfaces where data are presented in the form of flat-files. The following figure illustrates the components of the MV parser (shown inside the dotted line). It is a specific application of the translation framework shown in Figure 7. The parser is used in conjunction with a library of vendor independent mappings and some vendor specific mappings.

The multivendor Parser is shown in Figure 11. The MV Parser reads a ROF file (as defined above) and invokes the appropriate mapping for each of the resulting RCOs.

The structure of a ROF file, indeed almost all E/NMS data files, is such that an attempt at error recovery (for example, skipping until the next "class") could result in incorrectly set information. The advantages of error recovery are outweighed by the impact of reporting data inaccurately as a result of incorrect recovery.

The Parser is typically invoked from the command line, or scheduled to run at pre-determined intervals, and it is given the name of a file to be parsed. If the ROF file is successfully parsed the MV parser proceeds as follows;

1. The MV Parser constructs a RCO whose class name is *MappingStart*, and adds attributes for each of the attributes present in the MV parser configuration file, or passed on the command line. The MV parser then attempts to invoke a mapping named *MappingStart*.
2. The MV Parser performs a depth-first traversal of the RCOs such that each RCO is visited in the order in which it is first encountered in the input file. For each RCO the engine attempts to locate a C++ implementation of the mapping with the same name as the class name of the RCO. If no C++ implementation is found, an attempt is made to locate a Java implementation. If no implementation is found the engine skips to the next RCO.
3. Once a mapping has been loaded the MV Parser invokes the *runMapping* operation on the mapping object, passing as parameters the RCAs of the RCO in question.

4. Once all RCOs have been visited the MV Parser construct a RCO whose class name is *MappingEnd* and attempts to invoke a mapping of the same name.
5. The MV Parser exits.

It is normally acceptable for there to be no mapping of a given name. If a mapping is not found for a given RCO the MV parser proceeds to the next RCO. It is possible to request that an error message is emitted by changing a configuration option but this merely generates an error message; it does not prevent the MV parser from proceeding to the next RCO.

The following example illustrates the order of mapping execution;

```
ROFV1.0
class One {
        class Two {}
        class Three {
                class Four {}
        }
class Five {}
```

The mappings would be executed in the order MappingStart, One, Two, Three, Four, Five, MappingEnd.

The MV Parser accepts the following command line arguments. As with other Resolve™ modules, all values (except -ptag) can be set in a configuration file. If a command line argument/configuration parameter is set incorrectly the MV Parser will terminate immediately.

- -ptag <tag>

This flag is used in the same manner as other Resolve™ modules

- -file <filename>

Identifies the file to be parsed. This argument is required. Only one file may be specified.

- -check_only [true | false]

If set to true the parser only checks the input for validity as per section; no mappings are executed and no mapping engine is created. The parser terminates after printing a message that indicates whether or not the file was valid. This argument is optional. The default is false which means that the parser does not print such a message and proceeds to apply mappings as described in section.

- -version [true | false]

If set to true the parser prints its version number and the date it was last built. This argument is optional. The default is false.

- -start_mapping <name>
  -end_mapping <name>

The name of the start and end mappings. Both are optional. If omitted they default to *MappingStart* and *MappingEnd* respectively.

- -loadjava [true | false]

If set to the false then the Java Mapping Engine is not used, and therefore only C++ implementation of mappings will be permitted. This argument is optional. The default is true.

- -parse_failure [informational | serious | fatal | operator | none]

Defines the severity of the error message if an input file fails to parse. This argument is optional. The default is informational. If set to none, no error message will be emitted in the event of a parse failure.

- -missing_mapping [info | warning | serious | fatal | operator | none]

Defines the severity of the error message if a mapping is not located. This argument is optional. The default is none which indicates no error message will be emitted if a mapping cannot be located.

- -classpath <classpath>

This is used by the Java Virtual Machine (JVM). If it is not set uses the $CLASSPATH environment variable. If $CLASSPATH is not set it uses $RESOLVE™HOME/java. If $RESOLVE™HOME is not set that the JVM cannot be initialized, an error

message (severity *serious*) is logged and no Java mappings can be executed. This option is ignored if loadjava is set to false (see above).

- -dispatching_class <name>
  -dispatching_method <name>
  -dispatching_package <name>

The name of the dispatching class and dispatching method used to load Java classes. The classes are loaded from the package specified by -dispatching_package. These flags are optional and are intended to be used only by developers of the translation framework. Users of the framework would not normally use these flags. These options are ignored if loadjava is set to false (see above).

Note that by convention ROF files have a ".rof" extension, although this is not enforced by the parser.

The Multi-Vendor Event Collector is a particular application of the translation framework to the specific task of event collection.

Some E/NMSs emit event information in a flat-file form and therefore this information may be processed by the MV parser (with the appropriate mappings, of course). However it is more common to collect information through some sort of API, and thus an alternative approach is required.

Figure 12 illustrates the components that make up the MV Event Collector (inside the dotted line) and those that must be added to it (outside the dotted line);

The MV Event Collector contains fewer standard components than the MV Parser as the "south bound" interface is not known (contrasts with the parser where it is known to be a ROF file).

Recall that mappings act upon RCOs. RCOs are created by the MV parser, but it is possible to create them directly, without recourse to flat-file. It is therefore possible to utilize mappings and the translation framework for E/NMSs that present information in formats other than flat-file.

Event collectors can be started, stopped and requested to perform synchronization operations in a consistent manner. If an event collector "dies" a director process detects this and will restart it.

The MV Event Collector supports this same interaction (i.e RCI interface) ensuring that all new multi-vendor event collectors appear to the rest of Resolve™ (and hence to the users of Resolve™) to be indistinguishable from existing 46020 event collectors.

Upon start-up the MV Event Collector performs the following steps.

1. The MV EventCollector loads a vendor specific library and invokes a start function in the library. The intended behaviour of this function is to connect to the E/NMS and collect event related information. Upon receipt of some event information the vendor specific collector would typically create an RCO(with associated RCAs) and request that a mapping be invoked on the RCO.
   A prototypical vendor specific event collector function would be as follows;

```
begin
    connect to E/NMS
    if sync on start required then do sync end if
    while not stopped
        case getCollectorState()
        sync :
            do sync
        event :
            request event info
            create RCO
            add RCAs containing information from event info
            invoke mapping on RCO
        shutdown:
            stopped = true;
        end case
```

```
        end while
        disconnect from E/NMS
    end
```

Note that getCollectorState() is periodically called in the loop. This indicates whether the vendor specific event collector should be collecting events, synchronizing or shutting down. If the MV Event Collector is suspended then the getCollectorState() function is blocked until the MV Event Collector resumes. In the interests of efficiency the vendor specific event collector should not call this function too often, once every second or so is adequate.

2. In a separate thread the MV Event Collector listens for RCI commands. After receipt of a sync message the MV Event Collector will return 'sync' the next time getCollectorState() is called. Upon receipt of a suspend message the MV Event Collector will block the collector thread next time getCollectorState() is called. The collector thread will remained blocked until an unsuspend or shutdown message is received. Upon receipt of a shutdown message the MV Event Collector will ensure that the next call to getCollectorState() returns 'stopped'. The collector thread will be given up to 30 seconds to terminate, before it will be killed and the MV Event Collector exits.

The vendor specific event collector thread functions are clearly E/NMS specific. Such functions are written in C (or C++ with C-style external linkage) and are kept in a shared library. The MV Event Collector will dynamically load this shared library at run time.

The MV Event Collector accepts the following command line arguments. As with other Resolve™ modules, all values (except -ptag) can be set in a configuration file.

- -ptag <tag>

This flag is used in the same manner as other Resolve™ modules

- -pmode [pmsync| pmevent]

This is the same as for the 46020 Event Collectors. If omitted the mode defaults to pmevent.

- -version [true | false]

If set to true the MV Event Collector prints its version number and the date it was last built. This argument is optional. The default is false.

- -loadjava [true | false]

If set to the false then support for Java is disabled, and therefore only C++ implementation of mappings will be permitted. This argument is optional. The default is true.

- -library <shared library>

Identifies the shared library to be loaded that contains the implementation of the start, sync and stop functions. Shared libraries are loaded using dlopen and therefore follow the search path as defined for that command (see man dlopen for details). This argument is mandatory. If the argument is not specified or missing the MV Event Collector will shutdown immediately with an error.

- -startFunction <function name>

Identifies the name of the start function for the vendor specific event collector. The function must have the parameter list(RWBoolean, MV_EcProcManProxy*, CK_Config*) and return type int. The function is found using dlsym(sees man dlsym for details). This argument is mandatory and if it is missing or not found the MV Event Collector will shutdown immediately with an error.

- -classpath <classpath>

This is used by the Java Virtual Machine (JVM). If it is not set uses the $CLASSPATH environment variable. If $CLASSPATH is not set it uses $RESOLVE™HOME/java. If $RESOLVE™HOME is not set that the JVM cannot be initialized, an error message (severity *serious*) is logged and no Java mappings can be executed. This option is ignored if loadjava is set to false (see above).

- -dispatching_class <name>
  -dispatching_method <name>
  -dispatching_package <name>

The name of the dispatching class and dispatching method used to load Java classes. The classes are loaded from the package specified by -dispatching_package. These flags are intended to be used only by developers of the translation framework. Users of the framework would not normally use these flags.

These options are ignored if loadjava is set to false (see above).

Specific Example

Statistics are currently imported into the Resolve™ HIB in a format know as ckload. A MV statistics importer is a translator that can convert from a the E/NMS format into ckload format.

In keeping with the two stage process the E/NMS data is first translated to a ROF file (syntactic translation) and then some mappings (semantic translation) applied to generate ckload files.

Suppose the E/NMS data were as follows;

3,123,23,11,22

Suppose further that the fields refer to the nmsID, circuit termination point ID, number of transmitted bytes (of two types) and number of received bytes respectively. It would be trival (in this simple example) to translate that into a ROF equivalent. For example;

```
ROFV1.0
class  ckload {
        filename = SI_frctp_1998_01_12_2315;
}
class fr_ctp_stats {
        nmsID = 3;
        ctpid = 123;
        TxBytes_typeA = 23;
        TxBytes_typeB = 11;
        RxBytes = 22;
}
```

Now mappings might be written as follows;

- ckload

  The ckload mapping takes the value of the filename attribute and opens a file of that name.

- fr_ctp_stats

  The fr_ctp_stats mappings takes the values of its attributes and writes them in a ckload format to the file previously opened. Now the ckload format understands TxBytes, but not the concept of two different types. Thus the mapping would take the values of TxBytes_typeA and TxBytes_typeB and add them together and write out the result. This is a form of semantic translation.

The advantage of such techniques is that format of ckload files is fixed and therefore it is possible to define a set of standard mappings. Thus to translate a new vendor's statistics it is only necessary to translate to a target ROF format and the standard mapping will then perform the necessary translation. This reduces the task of developing a "Vendor X" statistics importer to building a syntax translator only.

In cases where syntax translation is not sufficient it is possible to extend the standard mappings by means of inheritance (for mappings are implemented in an OO language). For example, suppose that Cascade FR UNI statistics were identical in all respects to FR UNI statistics as understood by Resolve™, with the sole exception of the calculation of the entity ID. It would be possible to write a new mapping (e.g. cascade_fr_uni_stats) that inherited the behaviour of the standard mapping, but over-rode the translation of the entity ID attribute. The developer of this mapping does not have to re-state the standard translation, only where the required translation is different.

If an E/NMS makes synchronization and/or event information in the form of an ASCII file the MV Parser can be used to perform the necessary synchronization/event propagation.

Again, this requires the development of mapping objects that makes CORBA calls to the Service Information API. For example, the following excerpt from a ROF file suggests how event information could be recorded;

```
ROF 1.0
class avc_event {
        service_component_type = FR_PVC;
        nmsID = 101;
        name = pvc123;
        op_state = enabled;
}
class avc_event {
        service_component_type = FR_PVC;
        nmsID = 101;
        name = pvc124;
        admin_state = locked;
}
```

Given such a file it is trivial to develop a mapping that would use this information to make CORBA calls to set the administrative/operational state of the specified service component. Such a mapping would be applicable to all service component types and would therefore need only be developed once. (This is the advantage of the two-step approach to translation).

In cases where an E/NMS provides event information through an API, and not through a flat-file, the above approach outlined in B.2 is still applicable. Recall that mappings work on RCOs, and that RCOs may be created directly. Thus it would be possible to collect event information from a vendor specific event collector and then create an RCO. For example, an RCO *avc_event*, with RCAs called *service_component_type*, *nms_id*, will be created and mapped without a flat file;

```
ROFObject objectName("avc_event");
ROFAttribute *attribPtr=NULL;
while more_attributes
        switch (attribute_type)
```

```
        {
        case service_component_type:
                attribPtr = new ROFAttribute("service_component_type",attribute_value);
                break;
        case nms_id:
                attribPtr = new ROFAttribute("nms_id",attribute_value);
                break;
        }
        objectName.addAttribute(attribPtr);
end while
runMapping(objectName);
```

The following discussion outlines the steps that would typically be performed to configure the translation framework for a specific E/NMS. It also includes some recommendations on how this configuration should be approached.

Building a network interface involves two separate translation steps - syntax translation and semantic translation. Both steps pivot around the central notion of an intermediate (ROF-based) representation and therefore choosing this representation is very important.

The main factor that will govern the choice of intermediate representation is an awareness of the library of pre-defined mappings specified. These pre-defined mappings have been specifically designed to ease the task of building network interfaces and it is anticipated that a new network interface would take these mappings as a starting point and then only derive vendor specific differences. Each of these mappings defines the required intermediate format. For example, suppose that there is a mapping that accepts RCOs in the following form;

```
class dummy {
        dummyX = <number>;
        dummyY = <number>;
}
```

Suppose further that the semantics of this mapping are appropriate for a new network interface, except that there is an extra number to consider, Z. The new intermediate format might then be;

```
class dummyPlus {
        dummyX = <number>;
        dummyY = <number>;
        extraZ = <number>
}
```

By choosing this representation it is possible to implement a mapping for *dummyPlus* that will inherit the functionality of *dummy*. The *dummy* mapping will still work because the names of the attributes are preserved.
Thus in this example, re-use of a mapping determined the intermediate format.

If an existing mapping cannot be re-used then the choice of intermediate format becomes more open. However, the following guidelines should be considered;

- Avoid objects with a large number of attributes.

  The time taken to execute a mapping is influenced by the number of attributes (including inherited). An RCO with 30 attributes, say, would suggest that the object in question is too big .

- Remember that the MV parser runs to completion and then exits

  Therefore if some mappings need to build intermediate data structures, remember that they will be rebuilt every time the parser is executed.

The syntax translator takes information in a native format and translate it to the intermediate format.

The main goal of this syntax translator is to perform syntactical translation. More complex (semantic) transform is to be avoided. For the following reasons;

- Right tool for the right job

  The mappings were designed to perform semantic translation. Numerous tools exist for doing syntactical translations...

- Limit the impact of syntactic changes

  - It is common for two version of the same network management system to output the same information in slightly different format. By restricting the syntax translation to syntax only it is necessary to develop two syntax translators but keep only one semantic translator. Had the syntax translation taken on more semantic transformation it would be difficult to achieve this level of separation.

If the native format is obtained through some API then the implementation approach is largely determined by the API.

If the native format is some sort of file format then a wide variety of tools are available. Perl, awk, yacc/lex, etc., all provide reasonable approaches to performing syntactic translation.

The above described method permits an application, such as a service level management application, to interface with multiple vendor platforms without the need to tailor the underlying application to each vendor platform.

Claims:

1. A method of importing data into a data processing system from a third party platform collecting data from physical objects, comprising the steps of creating an interface object having a canonical form with a set of permissible extensions, mapping said canonical form to an underlying, externally invisible form existing in said data processing system, creating additional interface objects using said permissible extensions to derive new mappings for said additional objects, associating said interface objects with said physical objects, converting said data to be imported to said canonical form in said interface objects, and transferring said data through said mappings to said data processing system, whereby said data processing system is decoupled from said third party platform.

2. A method as claimed in claim 1, wherein said canonical form is an abstract data format that supports arbitrary attributes on arbitrary objects.

3. A method as claimed in claim 1 or 2, wherein said imported data is availability and performance information.

4. A method as claimed any one of claims 1 to 3, wherein a service model object is also exported to the third party management platform to generate alarms and/or trouble ticket information related to the object model.

5. A method as claimed in any one of claims 1 to 4, wherein a gateway application exchanges data between the system and the platform.

6. A method as claimed in claim 5, wherein said gateway application is run on said third party platform.

7. A method as claimed in claim 5 or 6, wherein said gateway interacts with the user platform using locally specified application programming interfaces.

8. A method as claimed in any one of claims 1 to 7, wherein said data processing system contains platform objects representing physical objects in said third party platform, and actions performed on said interface object are mapped onto said platform objects.

9. A method as claimed in claim 8, wherein said mapping of said actions into said platform objects is performed with a Turing complete language.

10. A method as claimed in claim 9, wherein said third party platform is a network management system and said platform objects are network objects.

11. A method as claimed in claim 5, wherein said service model object is exported as a flat file.

12. A method as claimed in any one of claims 1 to 12, wherein two forms of mapping are performed, namely syntactical and semantic.

13. A method as claimed in any one of claims 1 to 13, wherein said interface objects are created automatically as physical objects are added to the system.

14. A system for importing data into a data processing system from a third party platform collecting data from physical objects, comprising an interface object for each physical object, said interface object receiving said data and having a canonical form and having a set of permissible extensions mappable to an underlying, externally invisible form in said data processing system so as to decouple said data processing system from said third party platform and permit mappings for new interface objects to be derived as extensions of existing mappings.

15. A system as claimed in claim 14, wherein said canonical form is an abstract data format that supports arbitrary attributes on arbitrary objects.

16. A system as claimed in claim 14 or 15, wherein said imported data is availability and performance information.

17. A system as claimed any one of claims 14 to 16, further comprising a service model exported to the third party management platform to generate alarms and/or trouble ticket information related to the object model.

18. A system as claimed in any one of claims 14 to 17, further comprising a gateway application on said third party platform that exchanges data between the system and the platform.

19. A system as claimed in claim 18, comprising locally specified application programming interfaces interacting with said gateway platform.

20. A system as claimed in any one of claims 14 to 19, further comprising platform objects representing physical objects in said third party platform, and means for mapping actions performed on said interface object onto said platform objects.

21. A system as claimed in claim 20, wherein said mapping means is a Turing complete language.

22. A system as claimed in claim 20, wherein said third party platform is a network management system and said platform objects are network objects.

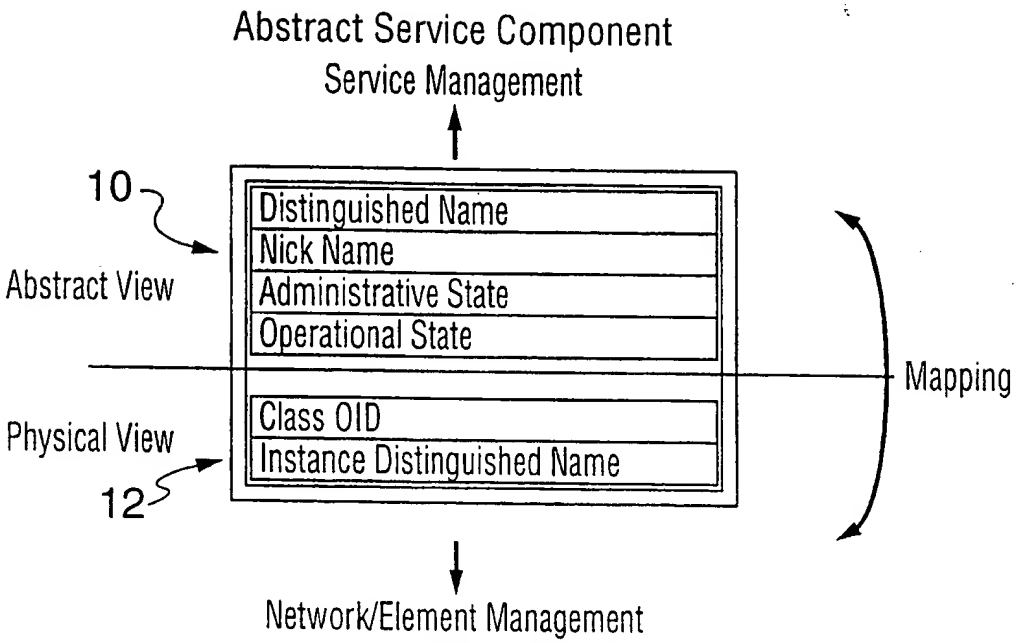

Abstract Service Component
Service Management
10
Distinguished Name
Nick Name
Administrative State
Operational State
Abstract View
Mapping
Physical View
Class OID
Instance Distinguished Name
12
Network/Element Management

FIG. 1

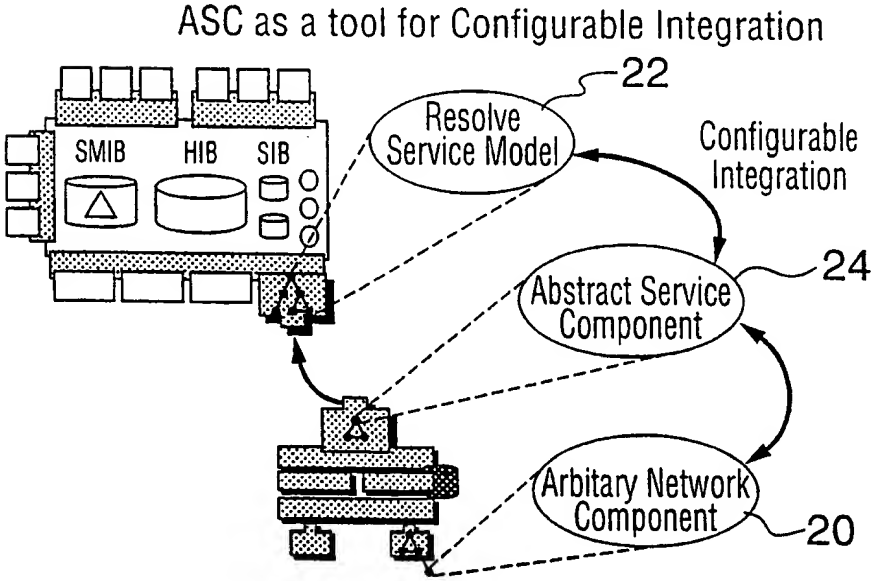

ASC as a tool for Configurable Integration
22
SMIB
HIB
SIB
Resolve Service Model
Configurable Integration
24
Abstract Service Component
Arbitary Network Component
20

FIG. 2

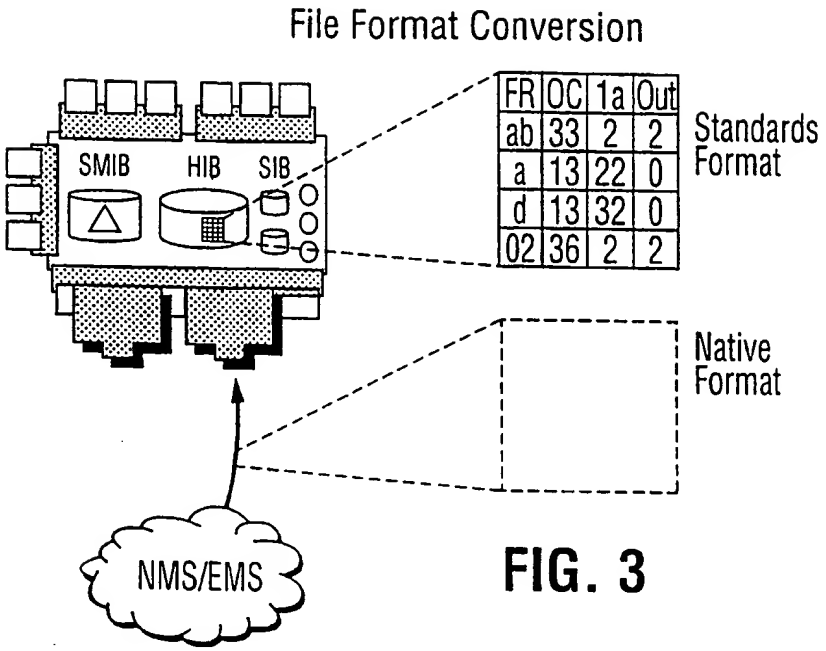

File Format Conversion
SMIB
HIB
SIB
FR OC 1a Out
ab 33 2 2
a 13 22 0
d 13 32 0
02 36 2 2
Standards Format
Native Format
NMS/EMS

FIG. 3

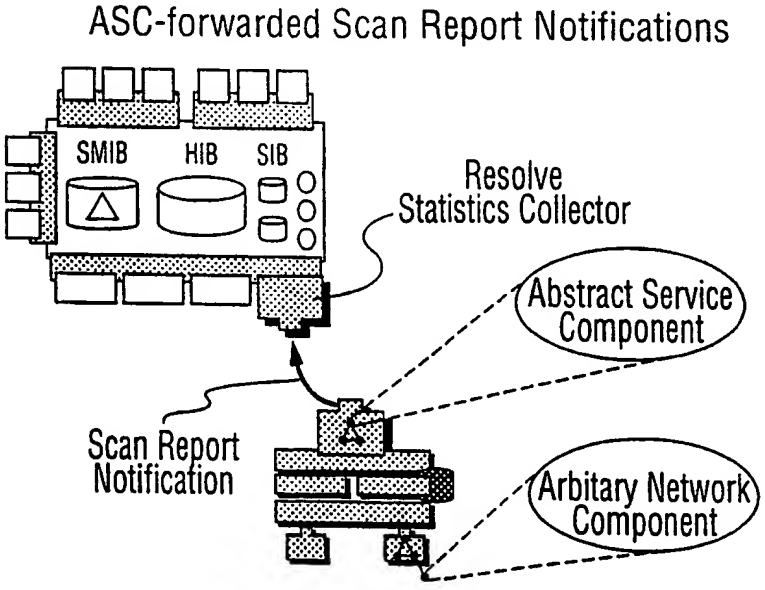

ASC-forwarded Scan Report Notifications
SMIB
HIB
SIB
Resolve Statistics Collector
Abstract Service Component
Scan Report Notification
Arbitary Network Component

FIG. 4

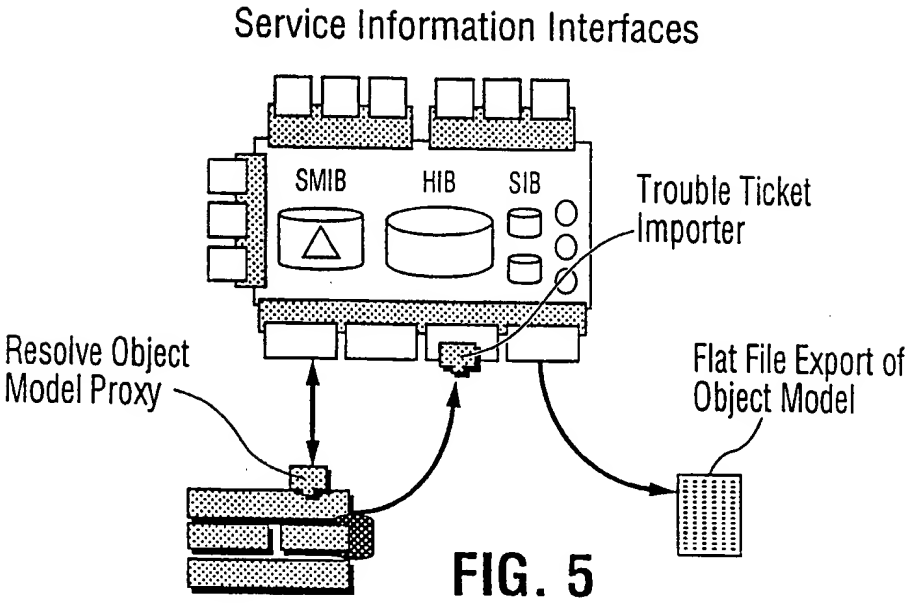

Service Information Interfaces
SMIB
HIB
SIB
Trouble Ticket Importer
Resolve Object Model Proxy
Flat File Export of Object Model

FIG. 5

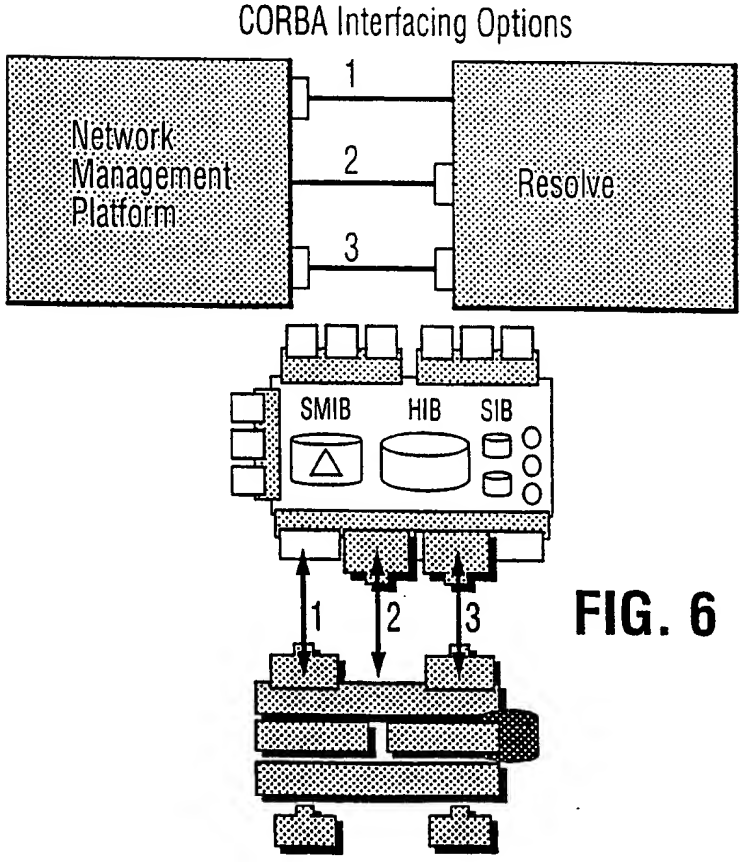

CORBA Interfacing Options
Network Management Platform
1
2
3
Resolve
SMIB
HIB
SIB
1
2
3

FIG. 6

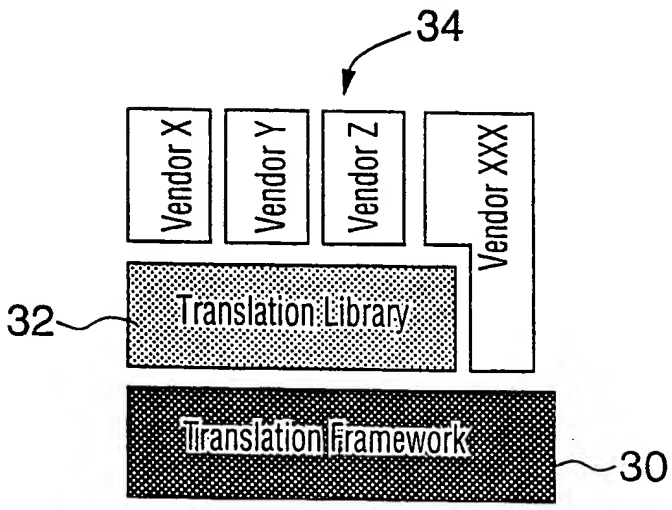

34
Vendor X
Vendor Y
Vendor Z
Vendor XXX
32
Translation Library
Translation Framework
30

FIG. 7

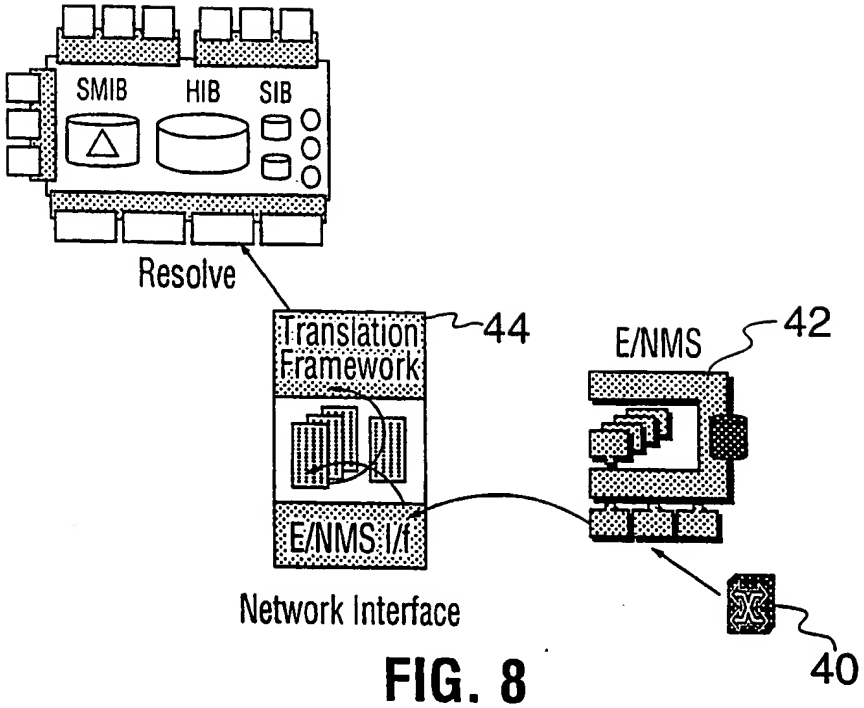

SMIB
HIB
SIB
Resolve
Translation Framework
44
E/NMS
42
E/NMS I/f
Network Interface
40

FIG. 8

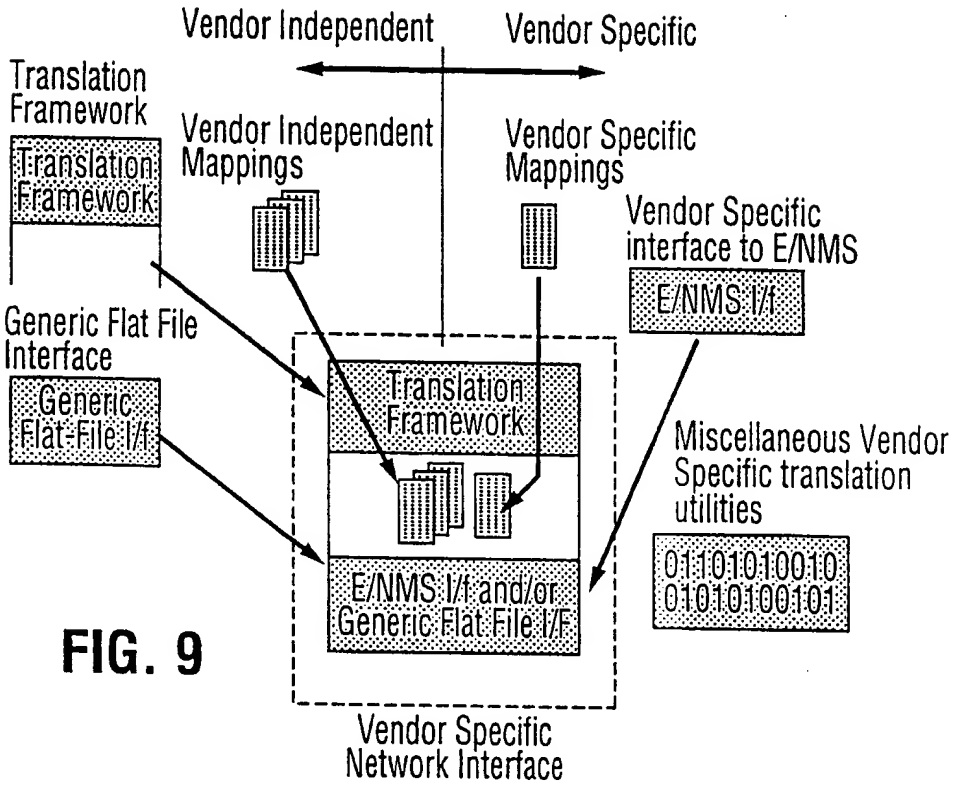

Vendor Independent
Vendor Specific
Translation Framework
Translation Framework
Vendor Independent Mappings
Vendor Specific Mappings
Vendor Specific interface to E/NMS
E/NMS I/f
Generic Flat File Interface
Generic Flat-File I/f
Translation Framework
Miscellaneous Vendor Specific translation utilities
0110101001010101010001 01
E/NMS I/f and/or Generic Flat File I/F
Vendor Specific Network Interface

FIG. 9

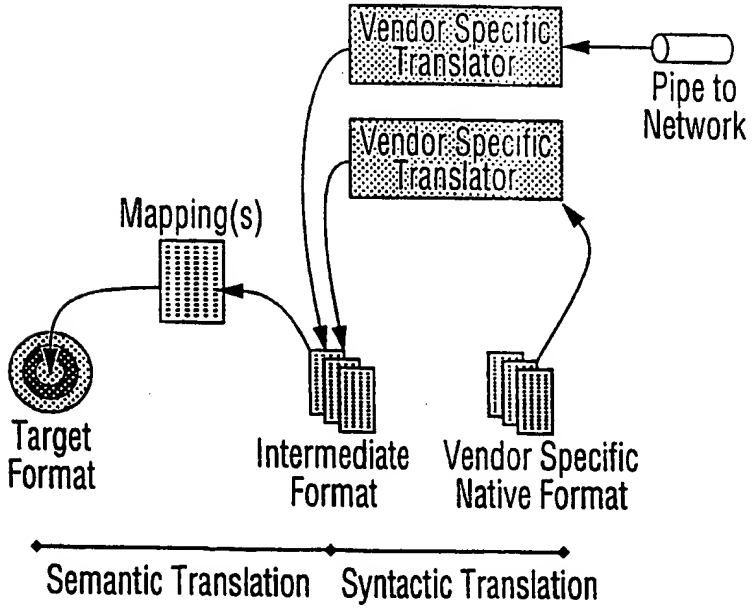

Vendor Specific Translator
Pipe to Network
Vendor Specific Translator
Mapping(s)
Target Format
Intermediate Format
Vendor Specific Native Format
Semantic Translation
Syntactic Translation

FIG. 10

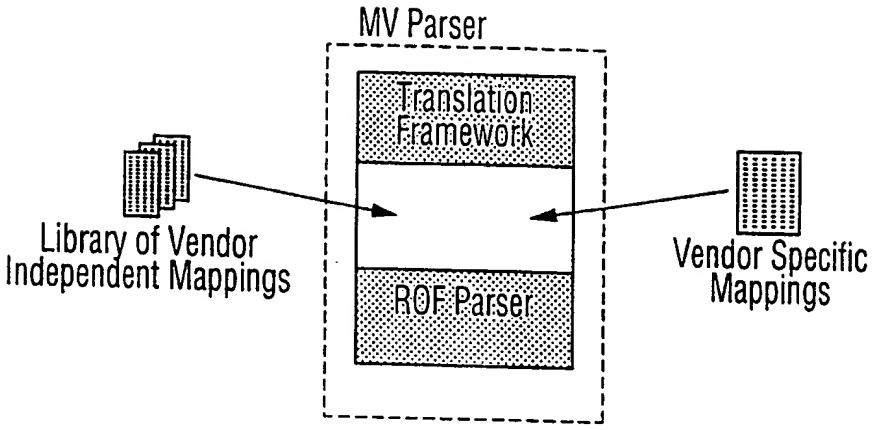

MV Parser
Translation Framework
ROF Parser
Library of Vendor Independent Mappings
Vendor Specific Mappings

FIG. 11

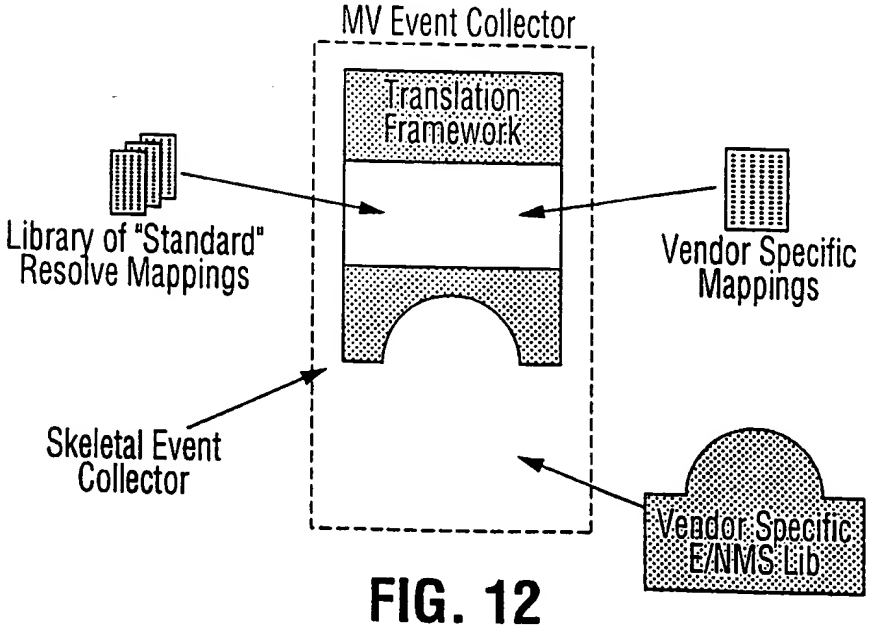

MV Event Collector
Translation Framework
Library of "Standard" Resolve Mappings
Vendor Specific Mappings
Skeletal Event Collector
Vendor Specific E/NMS Lib

FIG. 12

INTERNATIONAL SEARCH REPORT

Inte onal Application No
PCT/CA 98/00738

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04L12/24

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category '	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	NAZIM AGOULMINE ET AL: "A SYSTEM ARCHITECTURE FOR UPDATING MANAGEMENT INFORMATION IN HETEROGENEOUS NETWORKS" COMMUNICATION FOR GLOBAL USERS, INCLUDING A COMMUNICATIONS THEORY MINI CONFERENCE ORLANDO, DEC. 6 - 9, 1992, vol. 2, 6 December 1992, pages 999-1003, XP000357707 INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS	1,2,8, 12,14, 15,20
A	see the whole document	3-7, 9-11,13, 16-19, 21,22

-/--

[X] Further documents are listed in the continuation of box C.

[ ] Patent family members are listed in annex.

' Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publicationdate of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of theinternational search	Date of mailing of the international search report
28 October 1998	18/11/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Cichra, M

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inte. ...nal Application No
PCT/CA 98/00738

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication where appropriate, of the relevant passages	Relevant to claim No.
P,X	WEINSTOCK J ET AL: "An object-oriented approach to the management of distributed application systems" COMPUTER NETWORKS AND ISDN SYSTEMS, vol. 29, no. 16, 15 December 1997, page 1869-1879 XP004107249	1,14,20
A	see figures 1,2 see page 1872, column 1, line 13-19 see paragraph 2.1 see paragraph 2.4 see page 1877, column 2, line 7 - last line	2-13, 15-19, 21,22
A	TONY RICHARDSON: "STRATEGY FOR INTEGRATED MANAGEMENT SYSTEMS AND PLATFORM SUPPORT" IEEE NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM,14 April 1994, pages 545-554, XP000452357 see the whole document	1-22
A	HEINDEL L E ET AL: "TOTAL SYSTEMS MANAGEMENT IN OPEN ARCHITECTURE PROVISIONING SYSTEMS" PROCEEDINGS OF THE ANNUAL INTERNATIONAL PHOENIX CONFERENCE ON COMPUTERS AND COMMUNICATIONS, PHOENIX, APR. 12 - 15, 1994, no. CONF. 13, 12 April 1994, pages 393-398, XP000462585 INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS see the whole document	1-5,8, 12-18,20
A	HUNT R: "SNMP, SNMPV2 AND CMIP - THE TECHNOLOGIES FOR MULTIVENDOR NETWORK MANAGEMENT" COMPUTER COMMUNICATIONS, vol. 20, no. 2, March 1997, pages 73-88, XP000688489 see paragraph 7.2	1-22